Name: Juehao Lin

Course Number: EC601

09/17/2021

Analyzing Malicious Android Applications with Machine Learning

**Problem Statement**

With the number of mobile device users growing exponentially in the past decade [1], the need for mobile security has also increased. Mobile phone, while was defined as a device solely for call and receiving calls years ago, has evolved into a smarter device and more of a "personal" device. As a result, mobile security has become an inevitable aspect that will affect most users' experience.

We will be focus on the platform of Android OS in this project, as it is one of the most widely used mobile phone operation systems in the world [2]. In this project, we will be focus on analyzing Android OS applications and classifying distinguish malicious applications from benign applications. Such project may include the process of reverse engineering an apk package, extracting useful apk package features for SVM (support vector machine) training and classification based on Machine Learning model.

**Applications**

The result of our project can be used in detecting malicious Android OS applications from benign applications, which if used correctly, can offer protection to users' mobile devices, and prevent users from losing sensitive data to malicious attackers. In addition to that, if we are managed to automate our detection process and used the data to train our machine learning model, it can also help in determining identifiable patterns that can be used to prevent future malware infections and improve future identifications.

The societal significance of our impact is considered to be huge. As more and more sophisticated and advanced malware is being created, and malware has become one of the most severe threats to sensitive users' data nowadays, along with phishing sites [3]. As we stated above, mobile phone nowadays contains many user data considered to be sensitive (personal data revealing real-world identity, financial information, etc.), failure to protect user's device from malicious applications will not only result in the leak of user's information, it is also likely to result in financial loss due to the leak of financial information and scam as a result of the leak of user's data revealing user's real-world identity. Hence a reliable analysis of Android OS applications is important for nowadays mobile devices and user's experience, as well as user's financial security. It is already known that traditional signature-based solutions could not be considered effective in cases of unknown or modified patterns and consider to be a passive defense. Hence a different approach that uses static analysis to differentiate malicious from benign apps in the Android OS is deemed important for the constantly changing malware.

**Area of Focus**

Since the topic of analyzing malicious Android OS applications considered to be a huge topic that covers the topic of Reverse Engineering, Static/Dynamic Analysis, Code analysis, and even Machine Learning, we will be focus on Static Analysis of Android OS applications and build a classifier that distinguishes malicious from benign applications by training a Machine Learning model to do so. In order to train a Machine Learning model, extracting features from the apk package is also required as an input to our model, hence Features Extraction would be an area of focus in our project.

**Literature review**

Android, as a popular mobile OS, was first released in 2008. A few years later, security concerns were discussed with the increasing popularity of Android applications [4].

Android is built on top of the Linux Kernel. Linux is chosen because it is open source, verifies the pathway evidence, provides drivers and mechanisms for networking, and manages virtual memory, device power, and security [5]. Android has a layered architecture [6]. The layers are arranged from bottom to top. On top of the Linux Kernal Layer, the Hardware Abstraction Layer, Native C/C++ Libraries and Android Runtime, Java Application Programming Interface (API) Framework, and System Apps are stacked on top of each.

An android Malware usually relies on a user's Permissions to access on-device information. Every apk must include a manifest file that, among other things, requests permission to access certain restricted elements of the Android operating system. These elements include access to various hardware devices (e.g. GPS, camera), sensitive features of the operating system (e.g. contacts), and access to certain exposed parts of other applications [7]. Since most Android OS applications rely on permission to obtain the functionality, classifying standard built-in permissions and non-standard permissions is an important aspect of Static Analysis. By labeling standard built-in permissions and non-standard permissions into two different groups, such information can be feed into a Machine Learning model (such as SVM) to be used as a classifier to detect Malware.

In addition to permission, the manifest file also provides information about The app's package name. The Android build tools use this to determine the location of code entities when building projects. When packaging the app, the build tools replace this value with the Application ID from the Gradle build files, which is used as the unique app identifier on the system and on Google Play. Some level of malware detection can be done based on this information by comparing the identifier with the official database.

The components of the app, which include all activities, services, broadcast receivers, and content providers are also included in the manifest file. Each component must define basic properties such as the name of its Kotlin or Java class. It can also declare capabilities such as which device configurations it can handle, and intent filters that describe how the component can be started. By labeling standard components of the app and unusual components of the app, such as suspicious services requirements and unusual intents, it can also be a set of features for the Machine Learning model.

Code analysis is another important aspect of Static Analysis. Code analysis includes the analysis of API calls, Information flow, Taint tracking, Opcodes, Native code, and Cleartext analysis [9]. Most of these analyses will require some degree of reverse engineering. Some successful analysis based on code analysis has been successful in the industry. For example, DroidNative has been introduced by Alam et al. as a solution for detecting Android malware. This scheme considers both bytecode and native code analysis, and according to the authors, DroidNative is the first to build cross-platform (x86 and ARM) semantic-based signatures for Android [10].

System calls tracing is also a different yet effective approach in static analysis. Narayanan et al. [11] extract control flow graphs (CFGs) from APKs, and get rooted subgraphs from these CFGs by graph kernels. Then the neural network is leveraged to learn the distribution

representations of structural semantics from these rooted subgraphs. The experimental result shows this method outperforms the Android characteristic-based method.

**Open Source research**

There are many interesting open-source projects that touch on touch the topic of my area of interest. The most interesting and useful projects I found is listed below

**Androguard**

**License:** [**Apache-2.0 License**](https://github.com/androguard/androguard/blob/master/LICENCE-2.0)

One important tool use for analyzing apk package is Androguard [12]. Androguard is a python based tool, which can run on Linux/Windows/OSX, provided python is installed in the system. It can be used for reverse engineering Android apps. This entails taking the raw Android Package (.apk) files of the app and breaking them down to analysis, include extracting features for our Machine Learning model.

**Android Malware Dataset/APK Mirror**

**License: open collection**

To build our dataset, we will need malicious files sample to train and test out our model. We use malicious files sample from <https://github.com/ashishb/android-malware> as our malware dataset and use apk sample from <https://www.apkmirror.com/> as our goodware dataset.

TensorFlow

**License:** [**Apache-2.0 License**](https://github.com/tensorflow/tensorflow/blob/master/LICENSE)

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications.

TensorFlow was originally developed by researchers and engineers working on the Google Brain team within Google's Machine Intelligence Research organization to conduct machine learning and deep neural networks research. The system is general enough to be applicable in a wide variety of other domains, as well.

In our project, we can use TensorFlow to construct and train out Machine Learning model, which will be the classifier to classify android malware.

**Results**

We were able to use the Androguard to do some preliminary examination on example APK. For example, we analyzed the provided example of app-prod-debug.apk provided by the Androguard.

We are able to obtain the permission required by the apk by using the .get\_permissions() API:

apk.get\_permissions()

['android.permission.INTERNET',

'android.permission.WRITE\_EXTERNAL\_STORAGE',

'android.permission.ACCESS\_WIFI\_STATE',

'android.permission.ACCESS\_NETWORK\_STATE']

And obtain the list of all activites, which are defined in the AndroidManifest.xml:

apk.get\_activities()

['com.greenaddress.abcore.MainActivity',

'com.greenaddress.abcore.BitcoinConfEditActivity',

'com.greenaddress.abcore.AboutActivity',

'com.greenaddress.abcore.SettingsActivity',

'com.greenaddress.abcore.DownloadSettingsActivity',

'com.greenaddress.abcore.PeerActivity',

'com.greenaddress.abcore.ProgressActivity',

'com.greenaddress.abcore.LogActivity',

'com.greenaddress.abcore.ConsoleActivity',

'com.greenaddress.abcore.DownloadActivity']

There are still many API we can used for apk package analysis. But due to time limit we left those API for our future experiments.

[1] “Mobile Fact Sheet.” Pew Research Center: Internet, Science & Tech, 7 Apr. 2021, [www.pewresearch.org/internet/fact-sheet/mobile](http://www.pewresearch.org/internet/fact-sheet/mobile). [Accesses 09/16/2021]

[2] Statista. “Market Share of Mobile Operating Systems Worldwide 2012–2021.” Statista, 29 June 2021, [www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009](http://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009) . [Accesses 09/16/2021]

[3] Cook, Sam. “Malware Statistics and Facts for 2021.” Comparitech, 29 May 2021, www.comparitech.com/antivirus/malware-statistics-facts. [Accesses 09/17/2021]

[4] Mobile Operating System Market Share Worldwide. <https://gs.statcounter.com/os-market-share/mobile/worldwide/> . [Accesses 09/16/2021]

[5] Khan, J.; Shahzad, S. Android Architecture and Related Security Risks. Asian J. Technol. Manag. Res. [ISSN: 2249–0892] 2015, 5, 14–18.

[6] Platform Architecture. <https://developer.android.com/guide/platform> [Accesses 09/16/2021]

[7] Sahs, Justin, and Latifur Khan. "A machine learning approach to android malware detection." 2012 European Intelligence and Security Informatics Conference. IEEE, 2012.

[8] “App Manifest Overview” Android Developers, 15 Sept. 2021, <https://developer.android.com/guide/topics/manifest/manifest-intro>. [Accesses 09/18/2021]

[9] Kouliaridis V, Kambourakis G. A Comprehensive Survey on Machine Learning Techniques for Android Malware Detection. Information. 2021; 12(5):185. <https://doi.org/10.3390/info12050185> [Accesses 09/18/2021]

[10] Alam, S.; Qu, Z.; Riley, R.; Chen, Y.; Rastogi, V. DroidNative: Automating and optimizing detection of Android native code malware variants. Comput. Secur. 2017, 65, 230–246.

[11] ] A. Narayanan, G. Meng, L. Yang, J. Liu, and L. Chen, ‘‘Contextual Weisfeiler–Lehman graph kernel for malware detection,’’ in Proc. Int. Joint Conf. Neural Netw. (IJCNN), Jul. 2016, pp. 4701–4708.

[12] Anthony, Geoffroy. “GitHub - Androguard/Androguard: Reverse Engineering, Malware and Goodware Analysis of Android Applications ... and More!” Androguard, 2019, <https://github.com/androguard/androguard> [Accesses 09/18/2021]